

# Modelli del Ciclo di Vita del Software: come si fanno o si dovrebbero fare i sistemi software

---

# Ciclo di Vita del Software (CVS)

---

- ◆ ***Software life cycle***: The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and sometimes, retirement phase.
- ◆ Note: These phases may overlap or be performed iteratively.
  - Standard IEEE 610.12-1990

# Ciclo di Vita del Software (CVS)

---

- ◆ *Software development cycle*: The period of time that begins with the decision to develop a software product and ends when the product is delivered. This cycle typically includes a requirements phase, design phase, implementation phase, test phase, and sometimes, installation and checkout phase.
- ◆ Contrast with *software life cycle*.
  - Notes: (1) The phases listed above may overlap or be performed iteratively, depending upon the software development approach used.
  - (2) This term is sometimes used to mean a longer period of time, either the period that ends when the software is no longer being enhanced by the developer, or the entire software life cycle

# Modelli di Ciclo di Vita del Software

---

Un **modello del ciclo di vita del software** (CVS) è una caratterizzazione descrittiva o prescrittiva di come un sistema software viene o dovrebbe essere sviluppato

*W. Scacchi - Encyclopedia of Software Engineering Vol II pag 860*

I **modelli di processo software** sono precise, formalizzate descrizioni di dettaglio delle attività degli oggetti, delle trasformazioni e degli eventi che includono strategie per realizzare, ottenere l'evoluzione del software

*NB molti autori (Ghezzi, Sommerville, Pfleeger ...) usano i termini processo di sviluppo del software e ciclo di vita del software come sinonimi.*

# Diverse tipologie di CVS

---

- ◆ Esistono vari modelli di CVS, nati (scoperti?) negli ultimi 40 anni
  - Waterfall (cascata)
  - Prototyping
  - Incremental delivery (approcci evolutivi)
  - Spiral model
- ◆ Molti aspetti influenzano la definizione del modello
  - specificità dell'organizzazione produttrice
  - know-how
  - area applicativa e particolare progetto
  - strumenti di supporto
  - diversi ruoli produttore/ committente

# Fasi di un CVS: una vista di alto livello

---

- ◆ *Definizione*: si occupa del *cosa*.
  - Determinazione dei requisiti, informazioni da elaborare, funzioni e prestazioni attese, comportamento del sistema, interfacce, vincoli progettuali, criteri di validazione
- ◆ *Sviluppo*: si occupa del *come*
  - Definizione del progetto, dell'architettura software, della strutturazione dei dati e delle interfacce e dei dettagli procedurali; traduzione del progetto nel linguaggio di programmazione; collaudi
- ◆ *Manutenzione*: si occupa delle *modifiche*
  - correzioni, adattamenti, miglioramenti, prevenzione

# Modelli a Cascata (Waterfall) - Royce '70

---

- ◆ Popolare negli anni '70
  - reazione al “code and fix” originario
  - ispirazione dall’industria manifatturiera
- ◆ Modello sequenziale lineare
  - progressione sequenziale (in cascata) di fasi, senza ricicli, al fine di meglio controllare tempi e costi
  - definisce e separa le varie fasi e attività del processo
    - » nullo (o minimo) overlap fra le fasi
  - uscite intermedie: semilavorati del processo (documentazione di tipo cartaceo, programmi)
    - » formalizzati in struttura e contenuti
  - consente un controllo dell’evoluzione del processo
    - » attività trasversali alle diverse fasi

# Modelli a Cascata: organizzazione sequenziale delle fasi

---

- ◆ ogni fase raccoglie un insieme di attività omogenee per metodi, tecnologie, skill del personale, etc.
- ◆ ogni fase è caratterizzata dalle attività (tasks), dai prodotti di tali attività (deliverables), dai controlli relativi (quality control measures)
- ◆ la fine di ogni fase è un punto rilevante del processo (milestone)
- ◆ i semilavorati output di una fase sono input alla fase successiva
- ◆ i prodotti di una fase vengono “congelati”, ovvero non sono più modificabili se non innescando un processo formale e sistematico di modifica

# Modelli a cascata e modelli di processo industriali

---

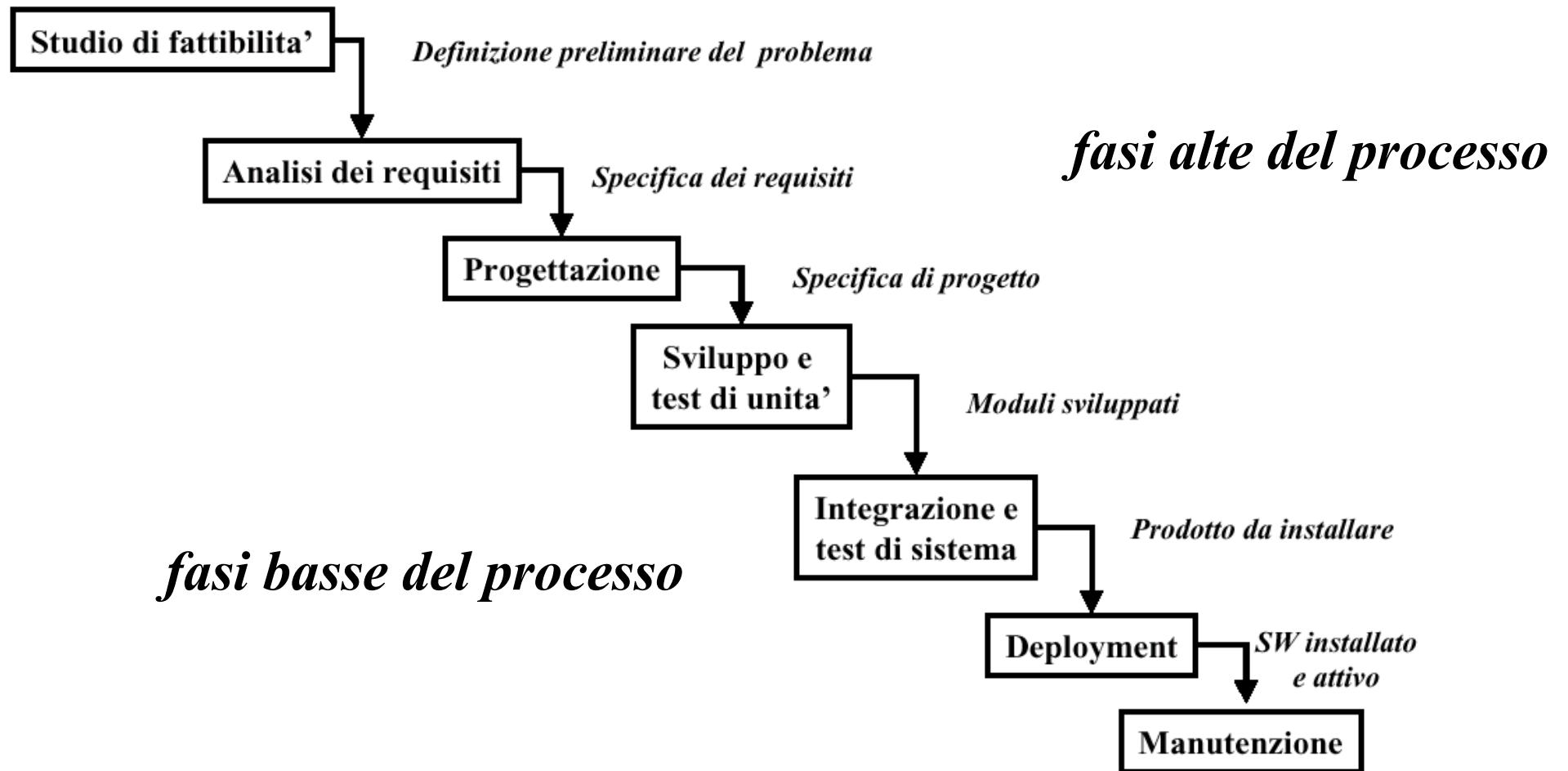
- ◆ Un esempio: costruzione di una casa ...
  - ... vorrei un casa su 2 piani, con autorimessa e cantina ...
  - ...prospetti, piantine,...
  - ...planimetrie, assonometrie,...
  - ...calcoli statici travi, infrastruttura elettrica, idrica,...
  - ...costruzione di pilastri, muri, impianto idrico...
  - ...verifica che il prodotto rispetti i desiderata del richiedente, nonché norme e/o standard, rilascio certificati di idoneità all'uso
  - ...il richiedente ci va a vivere...
  - ...dopo un po': riverniciatura delle pareti, riparazione del tetto, sopraelevazione, modifiche alla suddivisione in stanze, ...

# Modelli a cascata e modelli di processo industriali

---

- ◆ **Specifiche Requisiti** - definizione di requisiti e vincoli del sistema  
(vorrei un casa su 2 piani, con autorimessa e cantina...)
- ◆ **Progetto di Sistema** - produrre un modello cartaceo (planimetrie, assonometrie, ...)
- ◆ **Progetto di Dettaglio** - modelli dettagliati della parti (progetto dell'infrastruttura elettrica, idrica, calcoli statici travi...)
- ◆ **Costruzione** - realizzare il sistema
- ◆ **Test dei Componenti** - verifica le parti separatamente (impianto elettrico, idraulico, portata solai...)
- ◆ **Test di Integrazione** - integra le parti (il riscaldamento funziona...)
- ◆ **Test di Sistema** - verifica che il sistema rispetti le specifiche richieste
- ◆ **Installazione** - avvio e consegna del sistema ai clienti (ci vado a vivere)
- ◆ **Manutenzione** - (cambio la guarnizione al lavandino che perde ...)

# Modello a Cascata



# Studio di fattibilità

---

- ◆ Valutazione preliminare di costi e benefici
- ◆ Varia a seconda della relazione committente/produttore
- ◆ **Obiettivo**
  - Stabilire se avviare il progetto, individuare le possibili opzioni e le scelte più adeguate, valutare le risorse umane e finanziarie necessarie
- ◆ **Output** : documento di fattibilità
  - definizione preliminare del problema
  - scenari - strategie alternative di soluzione
  - costi, tempi, modalità di sviluppo per ogni alternativa

# Analisi dei requisiti

---

- ◆ Analisi completa dei bisogni dell'utente e dominio del problema
- ◆ Coinvolgimento di committente e ingegneri del SW
- ◆ **Obiettivo**
  - Descrivere le caratteristiche di qualità che l'applicazione deve soddisfare

**CHE COSA** 

 **COME**

- ◆ **Output:** documento di specifica dei requisiti
  - manuale d'utente
  - piano di *acceptance test* del sistema

# Progettazione

---

- ◆ Definizione di una struttura opportuna per il SW
- ◆ Scomposizione del sistema in componenti e moduli
  - allocazione delle funzionalità ai vari moduli
  - definizione delle relazioni fra i moduli
- ◆ Distinzione fra:
  - *architectural design*: struttura modulare complessiva (componenti)
  - *detailed design*: dettagli interni a ciascuna componente

## ◆ Obiettivo



- ◆ **Output:** documento di specifica di progetto
  - possibile l'uso di linguaggi/formalismi per la progettazione

# Fasi basse del processo

---

- ◆ **Programmazione e test di unità:** ogni modulo viene codificato nel linguaggio scelto e testato in isolamento
- ◆ **Integrazione e test di sistema**
  - composizione dei moduli nel sistema globale
  - verifica del corretto funzionamento del sistema
  - α-test: sistema rilasciato internamente al produttore
  - β -test: sistema rilasciato a pochi e selezionati utenti
- ◆ **Deployment:** distribuzione e gestione del software presso l'utenza
- ◆ **Manutenzione:** evoluzione del SW. Segue le esigenze dell'utenza. Comporta ulteriore sviluppo per cui racchiude in sé nuove iterazioni di tutte le precedenti fasi

# Modello a cascata: vantaggi e svantaggi

---

## ◆ Pro

- ha definito molti concetti utili (semilavorati, fasi ecc.)
- ha rappresentato un punto di partenza importante per lo studio dei processi SW
- facilmente comprensibile e applicabile

## ◆ Contro

- interazione con il committente solo all'inizio e alla fine
  - » requisiti congelati alla fine della fase di analisi
  - » requisiti utente spesso imprecisi: “l'utente sa quello che vuole solo quando lo vede”
  - » Errori nei requisiti scoperti solo alla fine del processo
- il nuovo sistema software diventa installabile solo quando è totalmente finito
  - » né l'utente né il management possono giudicare prima della fine dell'adesione del sistema alle proprie aspettative

# Nella realtà ...

---

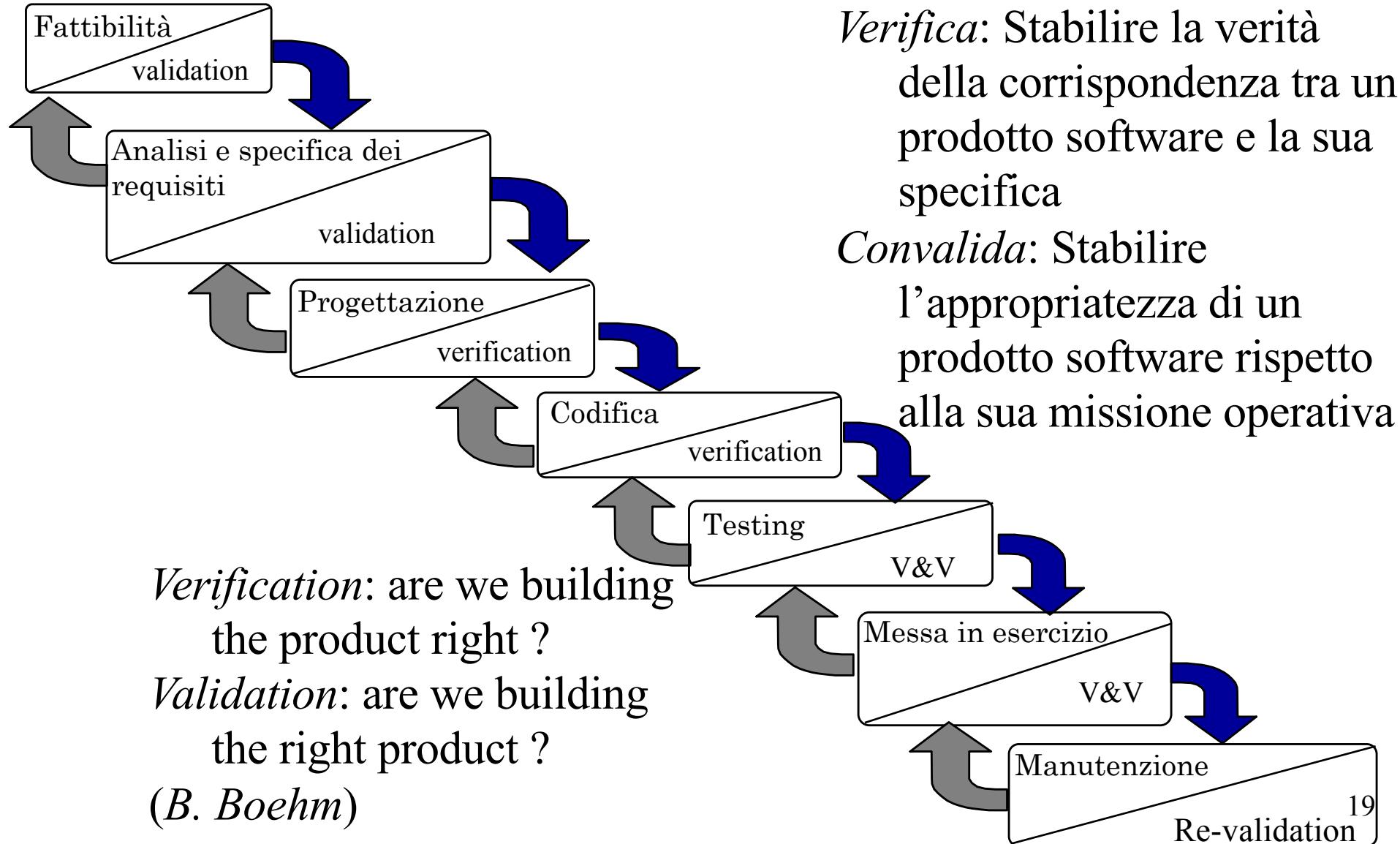
- ◆ Di norma le specifiche del prodotto sono incomplete o inconsistenti
- ◆ L'applicazione evolve durante tutte le fasi
  - Non esiste una netta separazione tra le fasi di specifica, progettazione e produzione
  - overlap e ricicli esistono!
  - in alcuni casi è auspicabile sviluppare prima una parte del sistema e poi completarlo (utente finale= mercato)
- ◆ Il software non si consuma e fare manutenzione non significa sostituire componenti
  - ... la manutenzione non può essere considerata marginale

# Altri cicli di vita

---

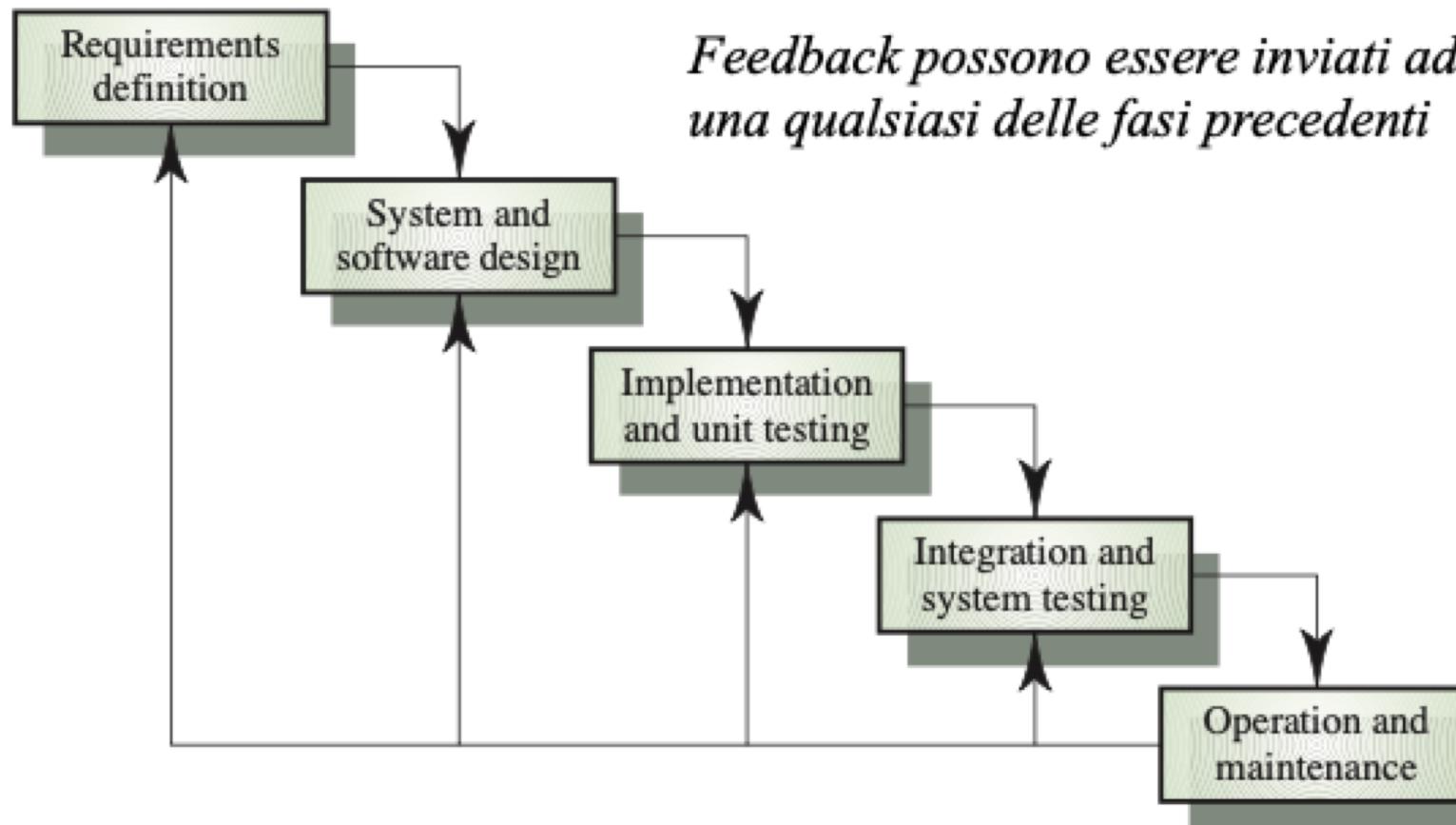
- ◆ Derivanti dalle critiche al modello a cascata
  - Vogliono ovviare al problema dell'instabilità dei requisiti
  - Tengono in considerazione l'esistenza dei ricicli
  - Maggiore enfasi sulla manutenzione (modelli evolutivi)
- ◆ Varianti del modello a cascata
- ◆ Modello trasformazionale
- ◆ Modelli con prototipo (“do it twice”)
- ◆ Modelli evolutivi ed incrementali
- ◆ Modelli basati su riuso
- ◆ Meta-modello a spirale

# Una variante del Waterfall Model: V&V e Retroazione (Feedback)



# ... Feedback

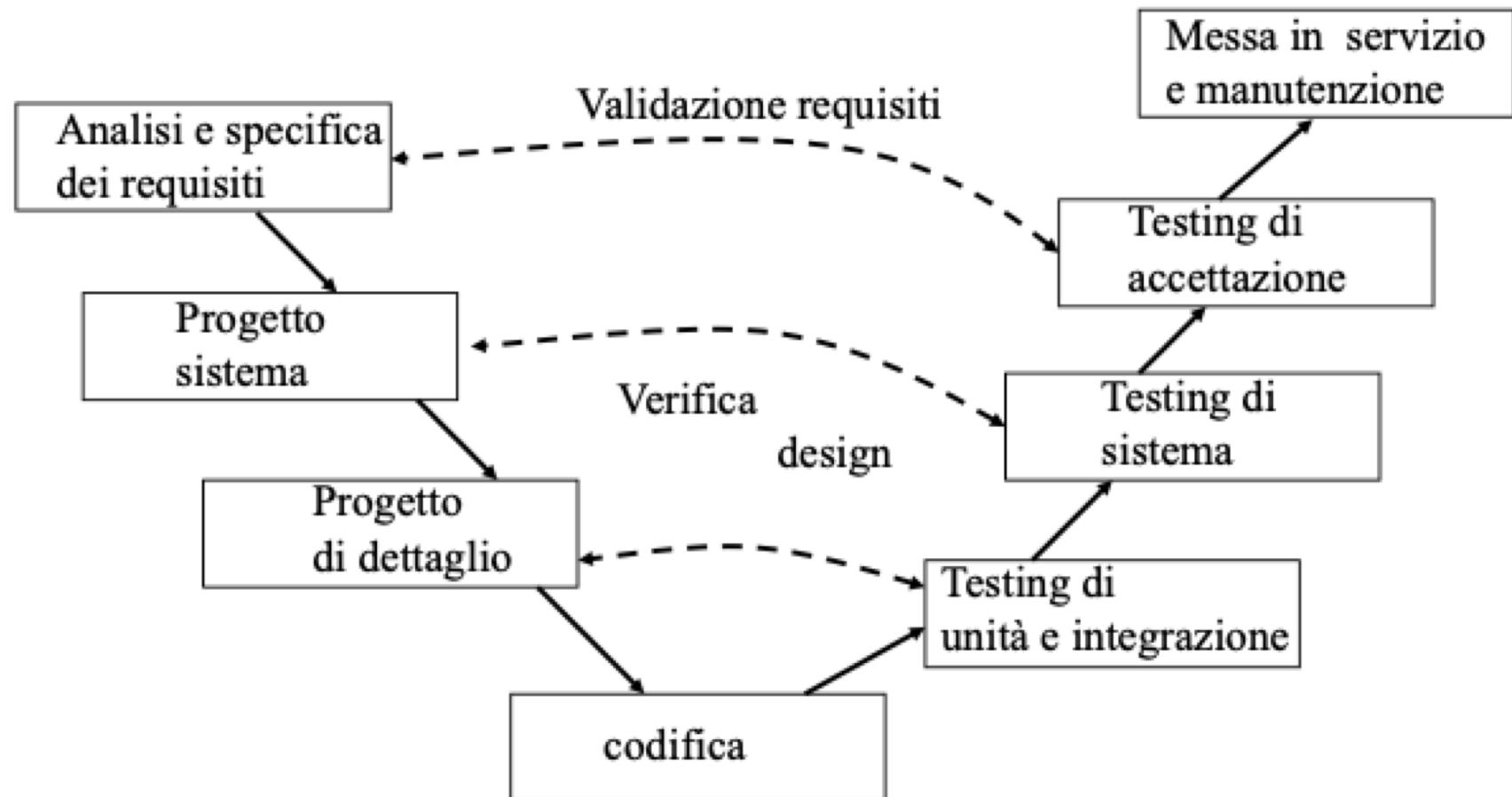
---



©Ian Sommerville

# Il modello a V

---



# Strategia del Modello a V

---

- ◆ Le attività di sinistra sono collegate a quelle di destra intorno alla codifica
- ◆ Se si trova un errore in una fase a destra (es. testing di sistema) si ri-esegue il pezzo della V collegato
- ◆ E' esplicito che si può iterare migliorando requisiti, progetto, codice .

# Modelli basati su prototipo

---

- ◆ Un *prototipo* per aiutare a comprendere i requisiti o per valutare la fattibilità di un approccio
- ◆ Realizzazione di una prima implementazione (prototipo), più o meno incompleta da considerare come una ‘prova’, con lo scopo di:
  - accettare la **fattibilità** del prodotto
  - validare i **requisiti**
- ◆ *il prototipo è un mezzo attraverso il quale si interagisce con il committente per accertarsi di aver ben compreso le sue richieste, per specificare meglio tali richieste, per valutare la fattibilità del prodotto*
- ◆ dopo la fase di utilizzo del prototipo si passa alla produzione della versione definitiva del Sistema SW mediante un modello<sup>23</sup> che, in generale, è di tipo waterfall

# Diversi tipi di Prototipazione

---

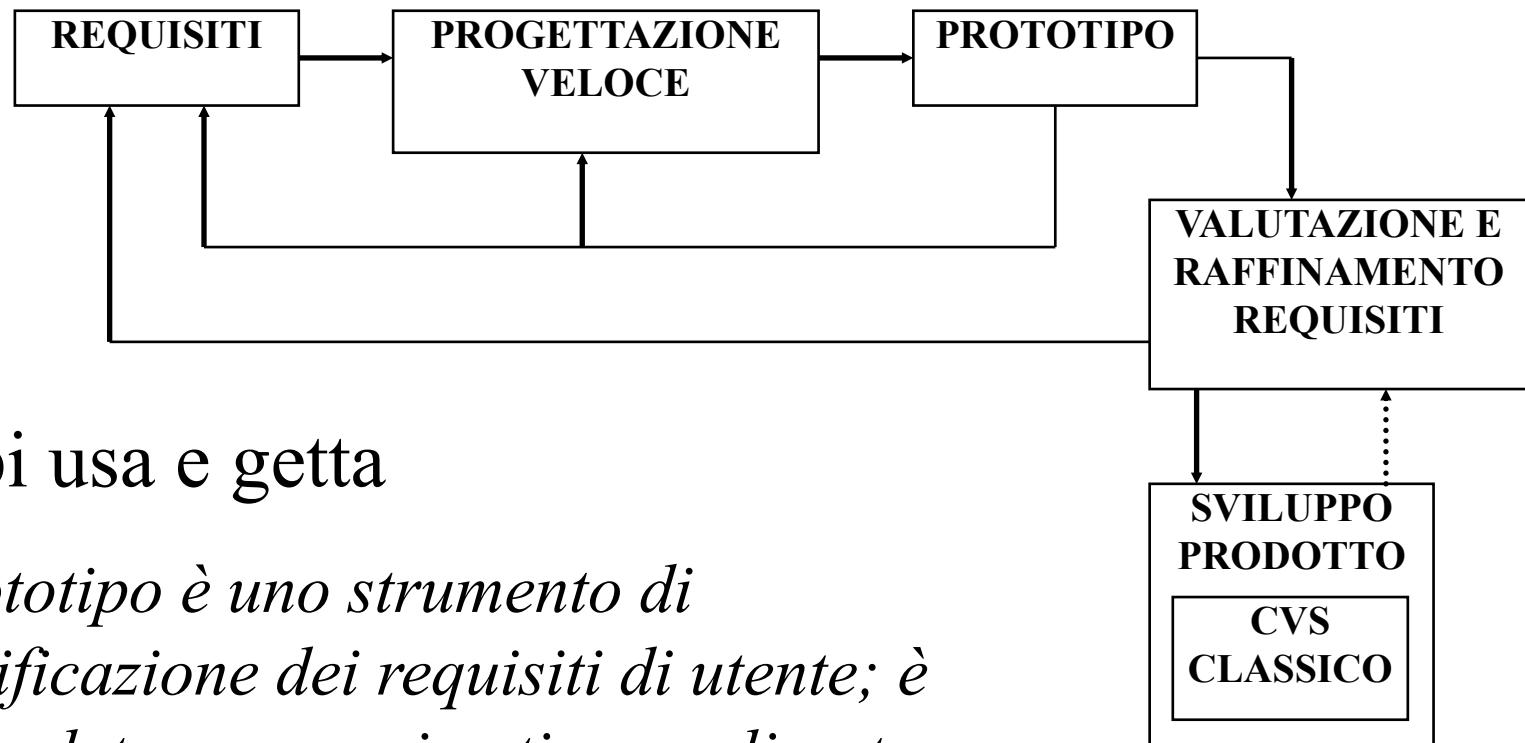
- ◆ **mock-ups**: produzione completa dell'**interfaccia utente**. Consente di definire con completezza e senza ambiguità i requisiti (si può, già in questa fase, definire il manuale di utente)
- ◆ **breadboards**: implementazione di sottoinsiemi di funzionalità critiche del sistema, non nel senso della fattibilità ma in quello dei **vincoli pesanti** che sono posti nel funzionamento del sistema (**carichi elevati, tempo di risposta, ...**), senza le interfacce utente. Produce feedbacks su come implementare la funzionalità (in pratica si cerca di conoscere prima di garantire).

# Prototipazione

---

- ◆ Prototipazione “throw-away”
  - Pervenire ad una migliore comprensione dei requisiti del prodotto da sviluppare
  - Lo sviluppo dovrebbe avviarsi con la parte dei requisiti **meno compresa**
- ◆ Prototipazione “esplorativa”
  - Pervenire ad un prodotto finale partendo da una descrizione di massima e lavorando a stretto contatto con il committente
  - Lo sviluppo dovrebbe avviarsi con la parte dei requisiti **meglio compresa**

# Modello con Prototipo Throw-away

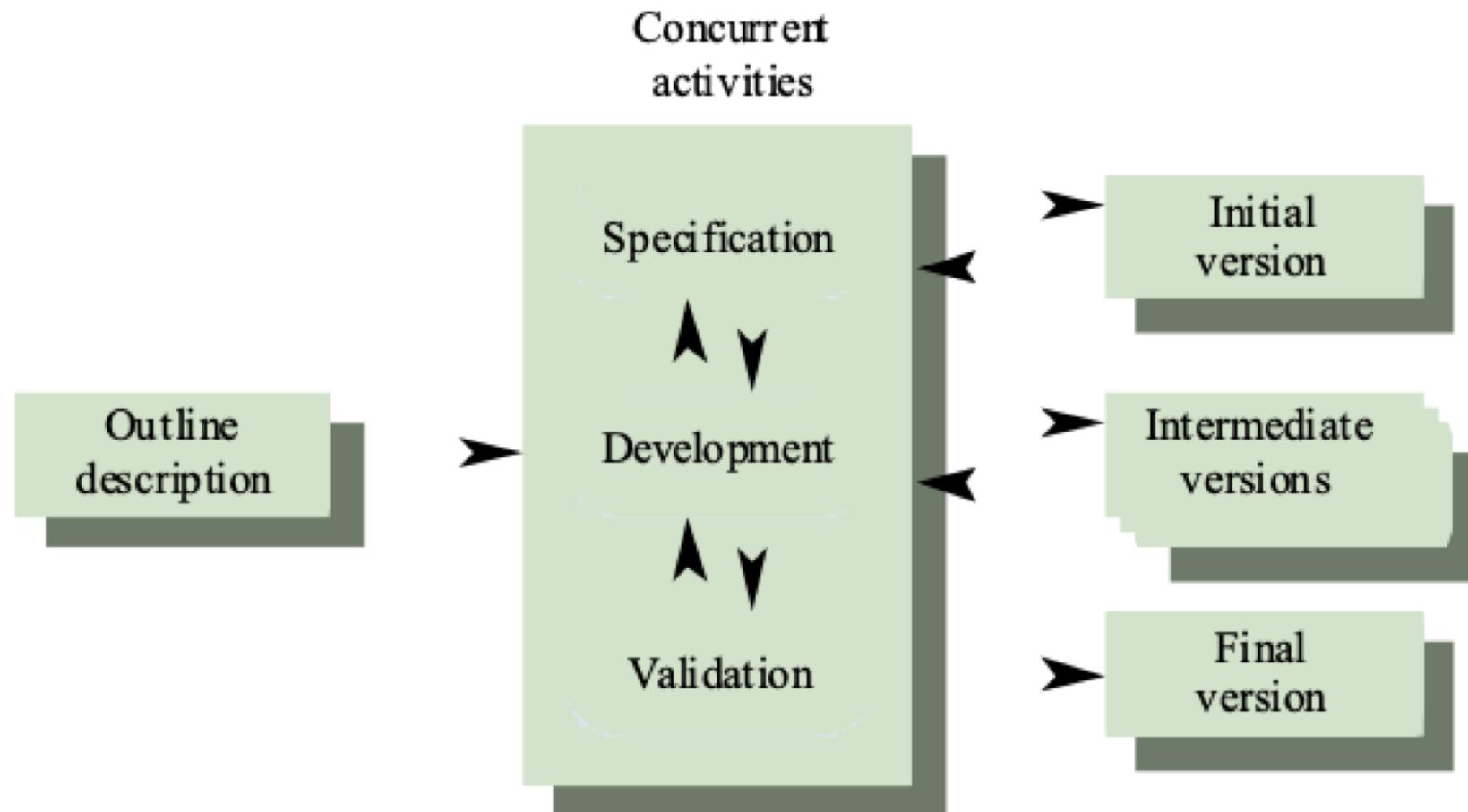


## ◆ Prototipi usa e getta

- *il prototipo è uno strumento di identificazione dei requisiti di utente; è incompleto, approssimativo, realizzato utilizzando parti già possedute*
- *Il prototipo **dove** essere gettato !*

# Sviluppo evolutivo

---



# Sviluppo evolutivo

---

## ◆ Problemi

- Perdita di visibilità del processo
- Spesso il prodotto finito è scarsamente strutturato
- Sono richieste competenze specifiche nell'uso di linguaggi di prototipazione rapida
- Perdita di visibilità del processo da parte del management

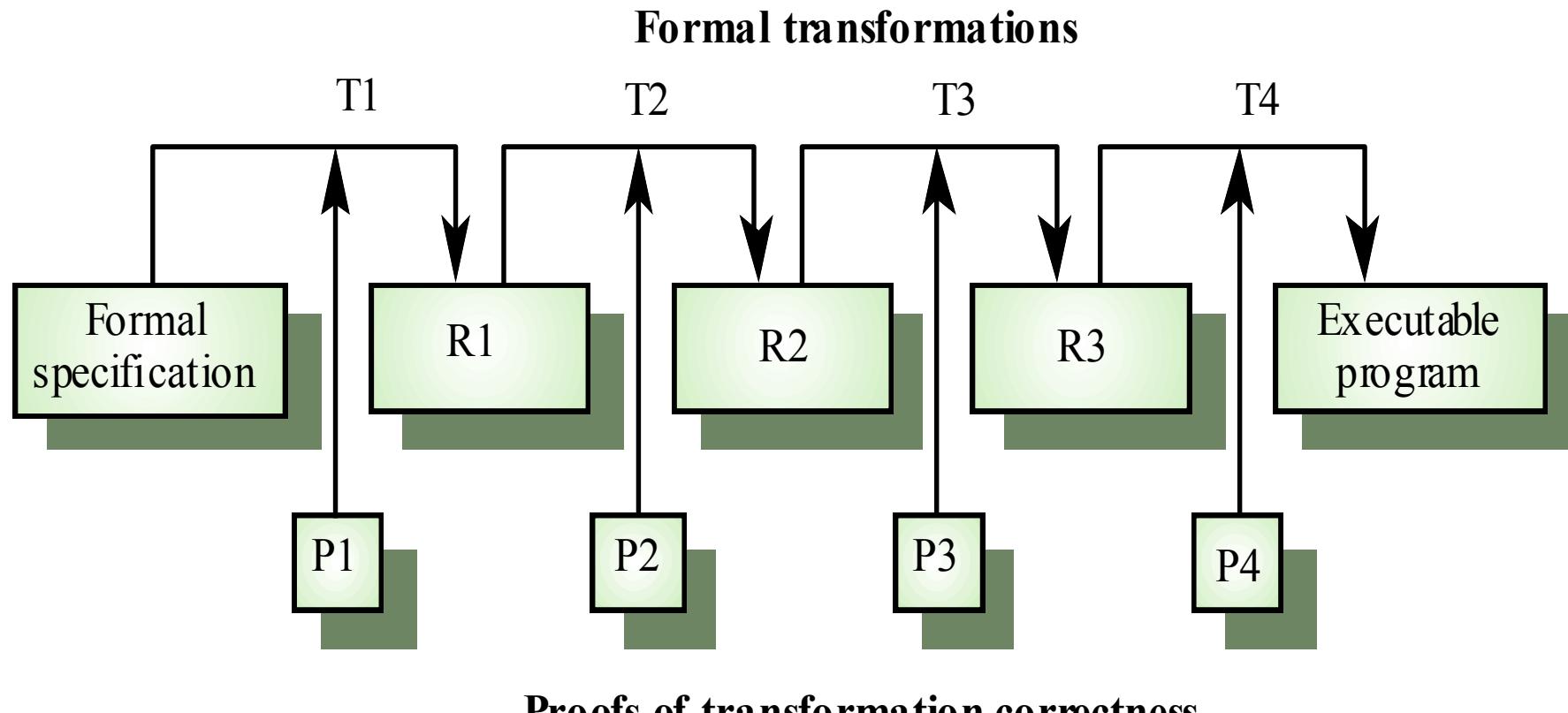
## ◆ Applicabilità

- Per sistemi interattivi di taglia medio-piccola
- Per parti di sistemi più grandi (esempio sviluppo UI)
- Per sistemi con un ciclo di vita breve

# Trasformazioni formali

---

*Lo sviluppo viene visto come una sequenza di passi che trasformano formalmente una specifica in una implementazione*



# Trasformazioni formali

---

- ◆ Problemi
  - Richiede competenze e skill specifici per l'applicazione delle tecniche
  - Difficoltà nello specificare formalmente alcuni aspetti del sistema (ad esempio l'interfaccia utente)
  - Pochi esempi di applicazione per sistemi complessi
  
- ◆ Applicabilità
  - Giustificata nello sviluppo di sistemi critici (safety o security)

# Modelli di sviluppo a componenti

---

- ◆ Basati su un riuso sistematico di componenti
  - sistemi software sono integrati da componenti esistenti o sistemi COTS (Commercial-off-the-shelf)
- ◆ Rapid application development
- ◆ Full Reuse Model (Basili, 1990)
  - prevede repository di componenti riusabili a diversi livelli di astrazione, prodotti durante le diverse fasi del ciclo di vita
    - » specifiche, progetti, codice, test case, ...
  - durante lo sviluppo di un nuovo sistema:
    - » riuso di componenti esistenti
    - » popolamento delle repository con nuove componenti
- ◆ Particolarmente adatti per sviluppo di software object-oriented

# Le iterazioni del processo

---

- ◆ I requisiti sono sempre soggetti a modifiche nel corso dello sviluppo. Questo comporta iterazioni di “rework” soprattutto nelle fasi iniziali
- ◆ Le iterazioni possono essere applicate a qualsiasi modello di processo di sviluppo
- ◆ Due approcci correlati:
  - Sviluppo incrementale
  - Sviluppo a spirale

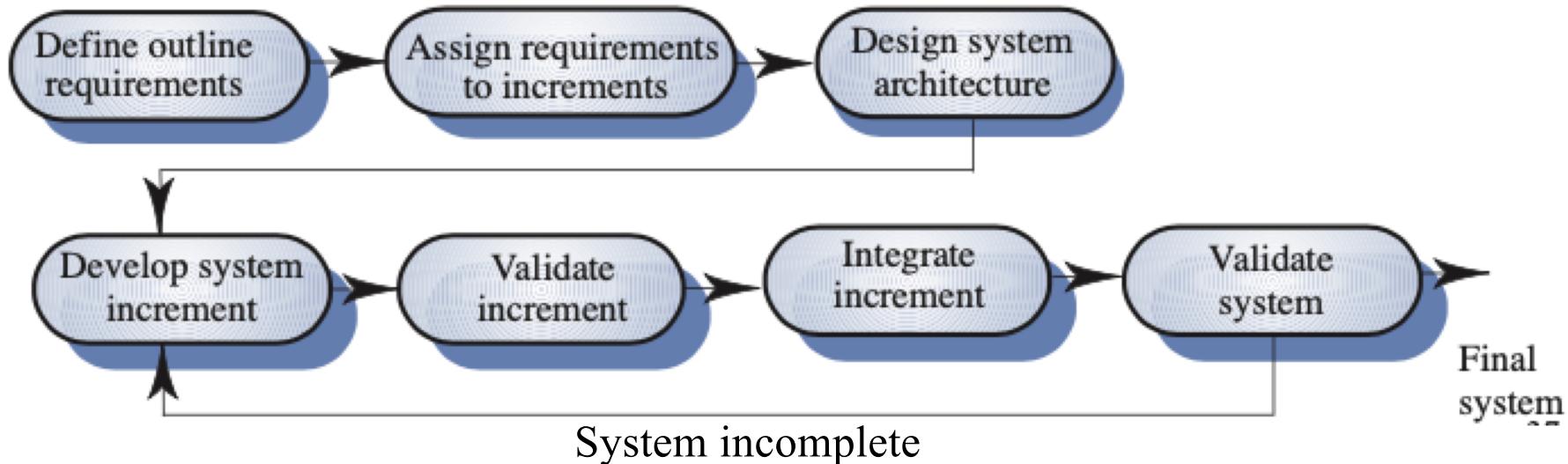
# Sviluppo “incrementale”

---

- ◆ Risolvono la difficoltà a produrre l'intero Sistema in una sola volta nel caso di grandi progetti SW
- ◆ Le difficoltà possono essere sia del produttore che del committente - quest'ultimo potrebbe non avere l'immediata disponibilità finanziaria necessaria per l'intero progetto
- ◆ Il prodotto viene consegnato con più rilasci

# Modello incrementale

- ◆ Le fasi alte del processo sono completamente realizzate.
- ◆ Il sistema così progettato viene decomposto in **sottosistemi** (incrementi) che vengono implementati, testati, rilasciati, installati e messi in manutenzione secondo un piano di priorità in tempi diversi.
- ◆ Diventa fondamentale la fase, o insieme di attività, di integrazione di nuovi sottosistemi prodotti con quelli già in esercizio



# Sviluppo incrementale (vantaggi)

---

- ◆ Possibilità di anticipare da subito delle funzionalità al committente
  - Ciascun incremento corrisponde al rilascio di una parte delle funzionalità
  - I requisiti a più alta priorità per il committente vengono rilasciati per prima
  - Minore rischio di un completo fallimento del progetto
- ◆ Testing più esaustivo
  - I rilasci iniziali agiscono come prototipi e consentono di individuare i requisiti per i successivi incrementi
  - I servizi a più alta priorità sono anche quelli che vengono maggiormente testati

# Modelli Incrementali - Modelli Iterativi

---

- ◆ Sono accomunati dal prevedere più versioni successive del sistema
- ◆ Ad ogni istante dopo il primo rilascio esiste un sistema versione N in esercizio ed un sistema N+1 in sviluppo
- ◆ Sviluppo incrementale
  - ogni versione aggiunge nuove funzionalità/sottosistemi
- ◆ Sviluppo iterativo (evolutivo)
  - da subito sono presenti tutte le funzionalità/sottosistemi che vengono successivamente raffinate, migliorate

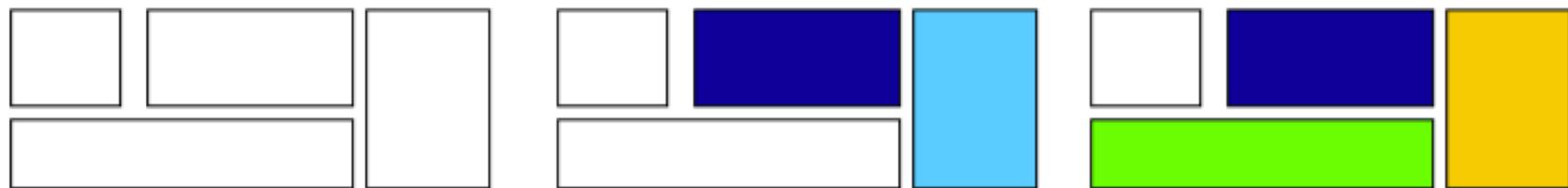
# Incrementale vs Iterativo

---

Incrementale



Iterativo



# Modello a spirale

---

- ◆ Formalizzazione del concetto di iterazione
  - Ha il riciclo come fondamento
- ◆ Il processo viene rappresentato come una spirale piuttosto che come sequenza di attività
  - Ogni giro della spirale rappresenta una fase del processo
  - Le fasi non sono predefinite ma vengono scelte in accordo al tipo di prodotto
- ◆ Ogni fase prevede la scoperta, la valutazione e il trattamento esplicito dei “rischi”
- ◆ E’ un meta-modello
  - Possibilità di utilizzare uno o più modelli

# Modello a spirale di Boehm

---

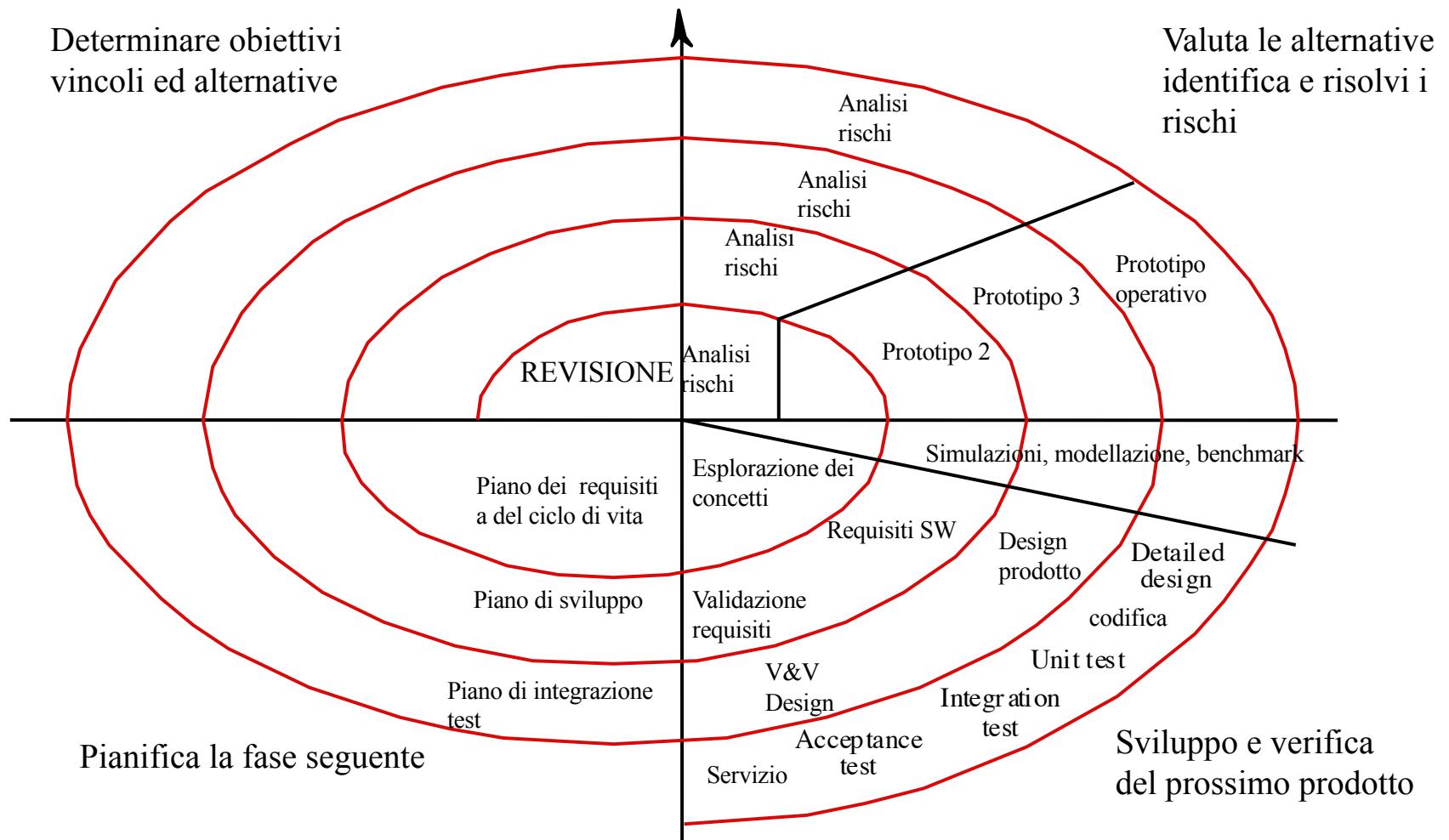


# Caratteristiche principali

---

- ◆ Consideration in each spiral cycle of the main spiral elements:
  - critical-stakeholder objectives and constraints
  - product and process alternatives
  - risk identification and resolution
  - stakeholder review
  - commitment to proceed

# Il modello a spirale tradizionale



# Gestione dei rischi

---

- ◆ Compito di chi gestisce (il manager) è minimizzare i rischi
- ◆ Tipologie di rischi: personale adeguato, scheduling, budget non realistico, sviluppo del sistema sbagliato ...
- ◆ Il rischio è insito in tutte le attività umane ed è una misura dell'incertezza sul risultato dell'attività
- ◆ Alti rischi provocano ritardi e costi imprevisti
- ◆ Il rischio è collegato alla quantità e qualità delle informazioni disponibili: meno informazione si ha più alti sono i rischi

# I modelli ed il rischio

---

- ◆ Cascata
  - alti rischi per sistemi nuovi, non familiari per problemi di specifica e progetto
  - Bassi rischi nello sviluppo di applicazioni familiari con tecnologie note
- ◆ Prototipazione
  - bassi rischi per le nuove applicazioni, specifica e sviluppo vanno di pari passo
  - alti rischi per la mancanza di un processo definito e visibile
- ◆ Trasformazionale
  - alti rischi per le tecnologie coinvolte e le professionalità richieste

# Modello a spirale

---

- ◆ L'articolazione di un progetto è guidata non da una rigida sequenza di fasi predefinite, ma da una gestione sistematica dei rischi di progetto, per arrivare alla loro progressiva diminuzione.
- ◆ All'inizio di un progetto di sviluppo software, i rischi sono tipicamente molto elevati.
  - Manca la chiarezza sui requisiti, le scelte sulle tecnologie e sulla strutturazione del sistema (le scelte architetturali) sono ipotesi non ancora consolidate.
  - In alcuni casi, sono state scelte tecnologie innovative, per le quali manca però una sufficiente esperienza nel gruppo di progetto.
  - In altri, anche a fronte di tecnologie conosciute, esistono incertezze legate alla necessità di fare fronte a un numero di utilizzatori contemporanei molto elevato, o a volumi di dati mai gestiti in precedenza.

# Modello a spirale

---

- ◆ Ogni iterazione ha lo scopo di ridurre i rischi di progetto.
- ◆ Inizialmente, tramite la costruzione di prototipi.
  - Prototipi di interazione (interfacce utente), per affrontare i rischi legati all'incertezza sui requisiti.
  - Prototipi architetturali (realizzazione e test di aspetti infrastrutturali), per affrontare i rischi legati alla scelta delle tecnologie ed i dubbi sulla strutturazione del sistema.
- ◆ Successivamente, quando i rischi principali sono stati messi sotto controllo, ogni iterazione ha lo scopo di costruire, in modo progressivo, nuove porzioni del sistema, via via integrate con le precedenti, e di verificarle con il committente e le altre parti interessate.
- ◆ Sotto questo profilo, esiste affinità con il processo di sviluppo incrementale; ma anche una differenza significativa.
  - Un processo iterativo, infatti, prevede una gestione sistematica del cambiamento di requisiti in corso d'opera. Prevede, in particolare, la "nascita" di nuovi requisiti espressi dal committente e dalle altre parti interessate al sistema come effetto dell'utilizzo del sistema stesso (delle sue parti già rese disponibili agli utilizzatori).

# Modello a Spirale e Flessibilità

---

- ◆ Per sistemi o sottosistemi di cui si ha una buona conoscenza si può adottare il modello a cascata: la fase di analisi dei rischi ha costi limitati
- ◆ Requisiti stabili, sistemi critici per le persone o cose (safety critical) possono essere sviluppati con approcci trasformazionali
- ◆ Zone non completamente specificate, interfacce d'utente possono impiegare il modello a prototipi

# Modello a spirale: vantaggi e svantaggi

---

## ◆ Vantaggi

- Rende esplicita la gestione dei rischi
- Aiuta a determinare errori nelle fasi iniziali
- Obbliga a considerare gli aspetti di qualità
- Integra sviluppo e manutenzione

## ◆ Svantaggi

- Per contratto di solito si specifica a priori il modello di processo e i “deliverables”. Lo sviluppo richiede un contratto nel contratto: vanno specificati vincoli, modello di processo, tempi di consegna e artefatti da consegnare
- La pianificazione dei progetti condotti in modo iterativo è più complessa. Il piano di un processo iterativo evolve durante tutta la durata del progetto stesso, e richiede un controllo sistematico degli avanzamenti.
- Un punto cruciale per il successo di un progetto iterativo è la collaborazione sistematica tra committenti (e altre parti interessate) e gruppo di progetto.
- Richiede persone in grado di valutare i rischi
- Per poter essere usato deve essere adattato alla realtà aziendale e/o al team

# Extreme programming

---

- ◆ Approccio recente allo sviluppo del software basato su **iterazioni** veloci che rilasciano piccoli incrementi delle funzionalità
- ◆ Partecipazione più attiva del committente al team di sviluppo
- ◆ Miglioramento costante e continuo del codice (verifica e adeguamento in tempi estremamente ridotti)

# Dodici regole dell'*Extreme Programming*:

---

- ◆ Progettare con il cliente;
- ◆ Test funzionali e unitari;
- ◆ Refactoring (riscrivere il codice senza alterarne le funzionalità esterne);
- ◆ Progettare al minimo;
- ◆ Descrivere il sistema con una metafora, anche per la descrizione formale;
- ◆ Proprietà del codice collettiva (contribuisce alla stesura chiunque sia coinvolto nel progetto);
- ◆ Scegliere ed utilizzare un preciso standard di scrittura del codice;
- ◆ Integrare continuamente i cambiamenti al codice;
- ◆ Il cliente deve essere presente e disponibile a verificare (sono consigliate riunioni settimanali);
- ◆ Open Workspace;
- ◆ 40 ore di lavoro settimanali;
- ◆ Pair Programming (due programmatori lavorano insieme su un solo computer).